Dynamic Event Driven Fault Diagnosis for Distributed Networks

Amit Kumar Gupta
&
Rigved Dattatraya Marathe

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

Dynamic Event Driven Fault Diagnosis for Distributed Networks

Thesis submitted in

May 2013

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Bachelor of Technology

by

Amit Kumar Gupta

(Roll 109CS0350)

and

Rigved Dattatraya Marathe

(Roll 109CS0587)

under the supervision of

Prof. Pabitra Mohan Khilar

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. www.nitrkl.ac.in

Dr. Pabitra Mohan Khilar
Assistant Professor

May 21, 2013

Certificate

This is to certify that the project work entitled in the thesis entitled *Dynamic Event Driven Fault Diagnosis for Distributed Networks* submitted by *Amit Kumar Gupta* and *Rigved Dattatraya Marathe*, bearing roll numbers 109CS0350 and 109CS0587 respectively is a record of original authentic work carried out by them under my supervision and able guidance in partial fulfillment of the requirements for the award of the degree of *Bachelor of Technology*in *Computer Science and Engineering* at National Institute of Technology, Rourkela.

To the best of my knowledge, neither this thesis nor the matter embodied in the project work has been submitted to any Institute/University for any Degree/Diploma/Academic award elsewhere.

Dr. Pabitra Mohan Khilar

Acknowledgment

On behalf of the submission of this thesis, we would like to express our respect and humble gratitude towards our supervisor Prof. P. M. Khilar, who guided and encouraged us throughout this project work. We would like to thank him for giving us deep idea in the field of Distributed System Level Fault Diagnosis and opportunity to work under his able guidance. We heartily appreciate the value of his guidance, encouragement and being source of inspiration to help us from beginning to the end of our research work. Without his invaluable advice and assistance it would not have been possible for us to complete this thesis. The experience what we got while working with such a wonderful person is simply wonderful.

Amit Kumar Gupta

Rigved Dattatraya Marathe

Abstract

The occurrence of faults is a common feature in most networks and addressing this issue is an important aspect of network maintenance. In this project, we take up the problem of detecting faults in general arbitrary networks. The networks that we have considered for the implementation of our algorithm are general arbitrary networks; where every node in the network has a minimum of two neighbors. Various types of faults occur in such networks. This algorithm is applicable to hard faults. While doing so, we have assumed our own set of well-defined assumptions which will be followed during the implementation phase of the algorithm. The approach taken to diagnose the system brings about a system level diagnosis and consists of two phases. Each of phases has its own complexity and constraints. In this algorithm we follow the principle of periodic testing that occurs after a certain fixed period has passed. During each such period the tester node performs a test on a fault free node and then determines whether the node it has tested is faulty or fault free. This is a brief abstraction of the first phase also known as the "Test Phase". The next Phase called as the "Circulation Phase" involves the tester node passing on the information of the test it has performed. However this information it passes will occur only in the presence of an event occurring during the "Test Phase". We have proposed a change in the algorithm which improves a performance metric. This metric is "number of messages passed". With the proposed algorithm the number of messages passed reduces in several cases of algorithm execution. In the results section we have performed a test on 50 nodes chosen at random, with random events periodically introduced in the network. The results shown include classification of faulty and fault free nodes, the number of messages passed and the diagnostic latency. Then a comparison of the existing and new proposed algorithm is shown with respect to the number of messages passed, highlighting the effect of the change. This project work could be applied to many problems where determining the state of a system at every stage is important.

***Keywords*:faults, diagnose, latency, message.**

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

An important problem in distributed systems that are subject to component failures is the distributed diagnosis problem. In distributed diagnosis, each working node must maintain correct information about the status (working or failed) of each component in the system. Static fault situation, in which an event can only occur after the previous event has been fully diagnosed. In dynamic fault environment, the nodes may change their status during execution of the diagnosis procedure[12]. One of the goal of distributed diagnosis is to allow fault-free nodes to achieve diagnosis in the shortest possible interval of time and minimum number of messages being passed. A fault-free node performs its specified system computation and commutation tasks correctly, and it has a local notion of time. A fault-free node is assumed to know which nodes are its physical neighbors in the network[11]. In a Distributed System-level Diagnosis algorithm for Arbitrary Network, fault-free processors perform simple periodic tests on one another; when a fault is detected or a newly- repaired processor joins the network, this new information is disseminated throughout the network.

1.2 Literature Survey

In [1], the adaptive algorithm is used for general topology networks which performs distributed diagnosis. Firstly, to achieve strong connectivity of the testing graph tests are added locally by each node simultaneously, to minimize diagnostic latency. In the second step, redundant tests are discarded by maintaining diagnosis of the system. While diagnosing the system, algorithm can diagnose any number of failures provided the graph of fault-free nodes is not divided in connected components.

In [2], the algorithm has three phases: search phase and destroy phase to construct testing graph, and inform phase which distributes a consistent final testing assignment to all fault-free nodes in the graph. Search phase is performed to add the tests locally at each node in the graph which gives a strongly connected testing graph to grant correct diagnosis of the system. Since addition of tests locally at each node may lead to redundant tests, therefore second phase is executed which removes the redundant tests and finally gives a minimally strong connected testing graph.

In [3], introduced a new algorithm called DNC, checks which part of the network is connected and also checks the reachability of a node in the network. This algorithm works for general topology and has three phases: testing phase, dissemination phase and connectivity computation phase. Among nodes and links any of the node or link can either be fault free or faulty, and hence it allows link fault as well as node fault. If there is no reply over a single given link then tester is not able to determine either it is link failure or the node connected by that link is faulty. But if the tested node does not reply to tests run over all possible links to tester then the tester considers that the tested node is unreachable i.e. faulty.

In [4], Hi-Comp algorithm not only hierarchical, distributed but also at the same time comparison-based, allows the diagnosis of the system which can be represented by complete graph. Since in this algorithm, the tests are based on comparisons; it is not limited to crash faults. The test is performed by a fault free tester which send same task to two nodes (processors), in turns the tester gets the executed outputs by two tested nodes. After receiving the executed outputs the tester compares the

outputs. If these two output are same then the tester declares these two nodes as fault-free, in case both are not same i.e. produces a mismatch then the tester declares that at least one of these two tested nodes is faulty but cannot tell which one. It is proven that the diagnosability of the algorithm is (N-1) and the latency of the algorithm is log2N, where N is the total number of nodes in the system.

1.3 Objective

Our objective is to create a random network consisting of an arbitrary number of nodes. Then we wish to introduce events in this network. By events we refer to failure of node/repair of failed node. The introduction of these events will be random in nature, but care will be taken to ensure that only failed nodes can be repaired and only fault free nodes can fail. While doing so, we consider the added clause of system non-partionability, since the algorithm will not support its introduction, and complete diagnosis of the system will not be possible, although the algorithm will function correctly in each connected component. Our overall objective will be to reduce the number of messages passed in the system which is an important metric for judging the performance of such algorithms. Since we foresee a trade-off occurring between the minimizing of information latency and decreasing the number of messages passed, one of these will be selected for better system performance. Also if any other metric that may arises at a later stage in the algorithm development phase, we shall try to optimize it to increase system performance.

Chapter 2

Background

2.1 Fault Classification

Handling faults is a key challenge in building distributed systems as it consists of different components. There are two commonly known approaches to handle this problem: *Fault masking* and *fault detection*. The main purpose of Fault masking is to hide the symptoms of a limited number of faults, so that users get correct service in the presence of such faults [6,8], whereas fault detection aims at identifying the faulty components in distributed system, so that the faulty components can be isolated and repaired [7, 10]. Faults can be categorized in mainly four fault classes [9], which are as follows...

1. **Non-observable faults:** If the fault free nodes cannot even be sure that the system contains any faulty nodes then this type of faults come under Non-observable faults class denoted by F_{NO}. The fault detection problem cannot be solved for any fault class F with $F \bigcap F_{NO} \neq \phi$.

2. **Ambiguous faults:** When a fault instance is in this class, the fault free nodes know that a faulty node exists, but they cannot be sure that it is one of the nodes in system. The fault detection problem cannot be solved for any fault class F with $F \bigcap F_{AM} \neq \phi$.

3. **Omission faults:** F_{OM} is the class of omission faults. For executions in this class, the fault free nodes could infer that one of the nodes in the system is faulty if they knew all the facts, but the positive facts alone are not sufficient; that is, they would also have to know that some message was not sent or not received. Intuitively, this occurs when the nodes in the system refuse to send some message they are required to send.

4. **Commission faults:**F_{CO} is the class of commission faults. For executions in this class, the fault free nodes can infer that one of the nodes in the system is faulty using only positive facts. Intuitively, this occurs when the nodes in the system send some combination of messages they would never send in any correct execution.

Therefore, if the fault detection problem can be solved for a fault class F, then F $\subseteq F_{OM} \bigcup F_{CO}$ and hence there is a solution to the fault detection problem with agreement for the fault class $F_{OM} \bigcup F_{CO}$.

Other faults which come under fault class $F_{OM} \bigcup F_{CO}$ are as follows:

- Permanent fault: In this scenario a faulty component in the system remains faulty unless it is repaired by some external entity.

- Transient fault: In this case, a fault occurs for some time and then disappears.

- Value fault: Value fault occurs when a node sends erroneous and non-expectable value to another node.

- Timing fault: When a node does not send an acknowledgement to another node with a predefined time interval.

2.2 System Model

System Model is considered as general arbitrary network in which nodes are connected by point to point communication links, where each node has got the minimum of two neighbors. The system is represented by undirected graph G (V, E) where, V is the set of vertices (nodes) and E is the set of edges (links). Node i and node j is called neighbor to each other if there exits an edge which connects node i to node j. Node failures are considered, whereas, if a link fails then it is assumed that node has failed (timeout can occur due to either of node failure or link failure). A node can be in either of two states: faulty or fault free. Each fault-free node knows the status of other nodes, also knows who its neighbor is and who is not; and perform test periodically on exactly one of its neighbor to know the status. Fault-free node is also able to respond to the test performed by one of its fault-free neighbor. Each fault-free node test another neighbor node and wait for an acknowledgment from tested node within a certain predefined amount of time. A tested node is assumed to response to test if it is fault-free else timeout occurs. A function of delay determines the timeout period. When a fault is detected by tester node then it is disseminated by the tester to all of its fault-free neighbors and fault-free neighbors broadcast this event message to all their fault-free neighbors and so on. Once a node becomes faulty, it is not being tested by any fault-free neighbor unless it is repaired by some means of external entity. An event (a node can become faulty from fault-free or vice versa) can occur at any time. If a node rejoins the system, it is assumed that the repaired node regains all of its properties i.e. it knows who are its neighbor and who are not, but doesn't know the status of other nodes. If the network is partitioned due to a set of faulty node(s), the algorithm stops until the set of node(s) which causes the partition is/are repaired. A node can change its state any number of time during the execution of the algorithm.

2.3 Assumptions

When a link fails, it is considered that a node is failed. k-connected arbitrary networks with k=1 are not included. A fault-free node Performs periodic test on one other fault free node and every fault free node is thus tested by one other fault free node. Faulty node fails to respond to tests sent by fault-free nodes. After proven faulty, it does not receive any message in the network and it regains all attributes after it rejoins the network.

Chapter 3

Algorithm

3.1 Basic Terminology

The following basic terminology is used in the algorithm:

1. Fault Free Node

    - A node is fault free if there is no fault occurring in it. Such a node is responsible for performing periodic testing of other fault free nodes.
    - Thus we can conclude every fault free node tests one other fault free node and is in turn tested by one other fault free node.
    - A fault free node responds to a test by other fault free node. This response is provided by an acknowledgement that is sent by the receiver. The failure of the arrival of acknowledgement indicates the node has failed.
    - It also has a property where it can request a neighboring fault free node to become it's tester in case it realizes that no other node is testing it.

2. Faulty Node

    - A faulty node is unable to respond to tests sent by a fault free node. It times out on the acknowledgement thus indicating its failure.

- After the node is declared as faulty it is no longer tested by any other fault free node as long as it is repaired.
- A faulty node in a network does not receive any of the messages that are propagated in the network.
- It regains all of its physical attributes after it has been repaired and has rejoined the network. But these physical attributes do not include system state information.

3. Orphan node A node who doesn't have a tester or who's tester has failed is called as a orphan node. There are two types of orphans:

   - Fault-free orphan: It happens when a fault free node's tester has failed.

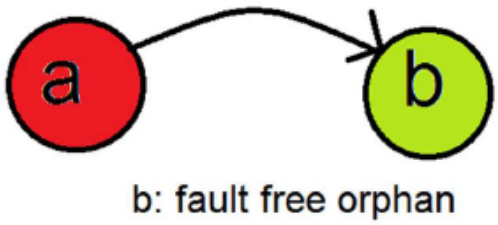


Figure 3.1: Fault-free orphan

   - Faulty orphan: Any node which has failed is a faulty orphan since no node is testing it by virtue of its failure.

3.2 Basic Phases

The algorithm is executed in two basic phases:

1. Test Phase
2. Circulation Phase

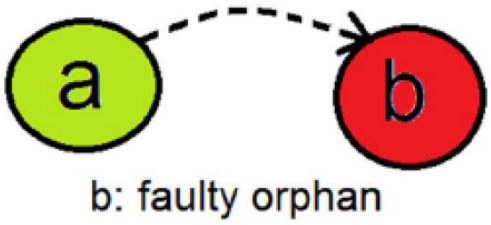


Figure 3.2: Faulty orphan

1. Test Phase: During this phase we basically detect a failure in a node. The tester node which is a fault free node periodically tests a node until there occurs a failure in that node. The test message is sent to the node via the channel and then a response is awaited for a certain time out period. If the time out occurs, that is no acknowledgement arrives from the node in that case the sender concludes that the node which it is testing has failed. Thus a failure has been detected in the network and it is up to the next phase to disseminate the information.

2. Circulation Phase: This is the second phase of algorithm execution phase and is the biggest phase too. There are plenty of steps in this phase many of which occur concurrently. After a tester node has detected a failure in node this phase resumes. In this phase the information of the failure is circulated throughout the network. First the tester node sends the information to its neighbors who in turn send the same to their neighbors until a point arrives when the entire network knows about the failure of the event. This phase is explained in much greater detail ahead.

3.3 Data Structures

We need to store and manipulate data to bring about the execution of the algorithm. For this purpose the following data structures at both node level and message level:

1. Local data: It is the data stored locally at each node in the network. When an event occurs, tester node first update the corresponding node's status locally. Each node also maintains the status about the neighboring node and non-neighboring node locally.

    - Event array: The event array is an N sized array where N is the number of nodes in the system. Every node has its own event array and is represented by $Event_j[1...N]$. The contents of this array are initially all zeros. If the event array has an Even value entry then we consider the node which is represented by the index of the array to be fault free. If the event array has an Odd value entry then we consider the node which is represented by the index of the array to be faulty. Every time an event occurs at a node the corresponding event value is incremented by 1. In the figure represented above we have considered the event array of Node 4. The green nodes also the fault free nodes are all represented by even values and the red nodes which are the faulty nodes are represented by odd values.

    - Neighbor Array: The neighbor array is used to keep vital information regarding the network for each node in the network. The contents of the neighbor array are shown as follows...

      In this figure the neighbor array for node 5 is shown. Here node 4 tests node 5 hence 2 is inserted at the index of node 4.Similarily Node 6 is tested by node 5 hence 1 is inserted in index of node 6. Node 1 isn't a neighboring node this 0 is present as its entry. Node 3 and Node 4 are neighbors but aren't testing node 5 thus their entry is 3 respectively.

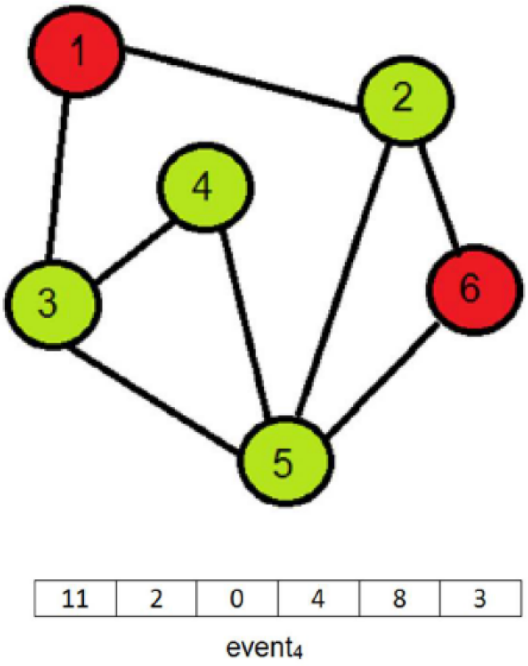


Figure 3.3: Event array of node 4

2. Message data: It consists of the data that is carried in through messages in the system.

   - Message Array: This array is represented by msg.event[1..N] and it is an N sized array where N is the number of nodes in the network. It contains the event values for each node in the network. When it arrives at a node it provides complete information regarding the system's state it carries. It carries even value for fault free node and odd value for faulty node in the network. It is generated by a node that has witnessed a node failure/node repair. In the figure below, the message composed carries Odd and Even values respectively.

     The message is generated by Node 1 which has detected a failure in Node 2 via periodic testing. Node 1 is also aware of Node 6 and Node 8's failure. It creates a message and sends it to Nodes 4 and 5. The Contents

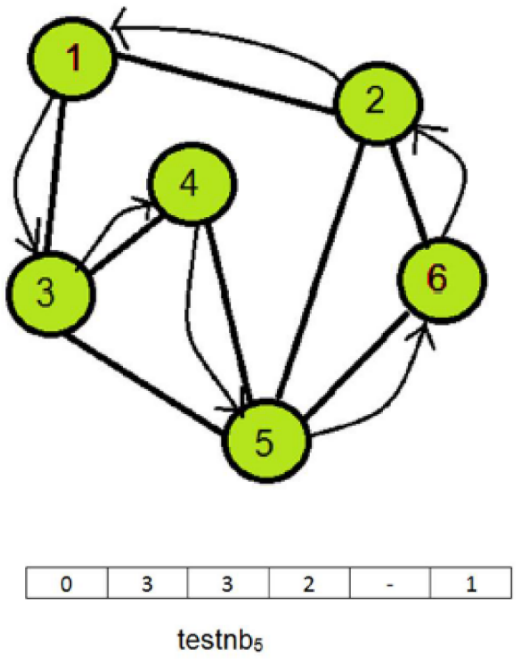


Figure 3.4: Test neighbor array of node 5

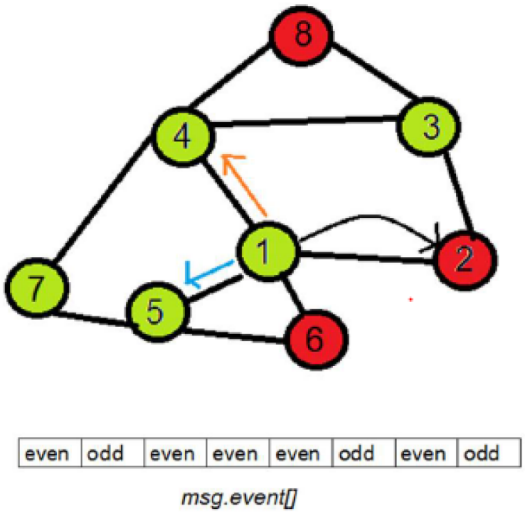


Figure 3.5: message array

of the message are even values for Nodes 1, 3, 4, 5 and 7. The message also contains Odd values for Nodes 2, 6 and 8.

- Intra-path Array: This data structure is used to prevent a message from going to the same node twice. In this case the sender of the message creates this N sized array where N is the number of nodes in the system. As the message traverses a node, the number 1 is inserted in its index of the corresponding array. Thus when a message appears at a node, the node knows which of its neighbors to send the message to. The presence of 0 in the index corresponding to a node indicates that node hasn't received the message yet. In the example above, the message that reaches node 5

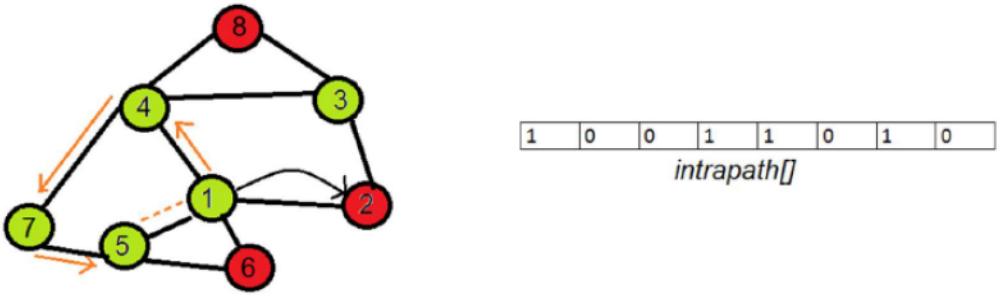


Figure 3.6: Intrapath array

will not from node 7 will not go back to node 1 since Node 1's intra-path value is already set to 1.

3.4 Secondary terminology

1. Testme: It is the request sent by a fault free orphan to one of its fault free neighbors when it realizes it has no tester node. This request is sent at random, however this has been modified in the proposed work later stated.

2. Tmj: It is the request sent by a node that rejoins the network after its failure. Since a newly joined node has no one to test it, this request is sent to one of its neighbors in order to obtain a tester. This request is sent to a neighbor at random whether it be faulty or fault free since a newly joined node has no

knowledge of which node is faulty and which is fault free.

3. To(acktm): It is the time-out message that occurs if a node which has received a testme request from a fault free orphan does not respond in a fixed time-out interval.

4. To(acktmj): It is time-out message that occurs if a node which has received a tmj request fails to respond in a fixed time-out period.

3.5 Detailed Explanation of Algorithm Phases

1. Detection Phase: When a tester node j detects a fault in a node i during the periodic testing phase it performs the following actions:

   - Increment eventj[i] by 1, to show the occurrence of fault.
   - Set testnb_j[i] to 3, which implies j and i are merely neighbors. This is the only processing that occurs in the detection phase of the algorithm.

2. Circulation phase In this phase, there are 3 sub-phases that bring about the actions:

   Sub-phase 1: In this phase we consider a node j has tested node i to be faulty. It then does the following processing in order to create and send a message to its fault free neighbors.

   - It creates a message that contains information of node i's failure. The message is loaded with the contents of its Event array as follows, msg.event[i] = event_j[i], $\forall$ i, $1 \leq i \leq N$.
   - The intra-path array is set to reflect that the message was created at node j as follows, intra-path[j]=1. All other entries in the intra-path array are set to 0.

- The message is then sent to all fault free neighbors of node j which is accomplished via a send-transaction() message as follows, $\forall$ k — $event_j[k]$ is even and $testnb_j[k]$ = 1,2, 3, or 4: send-transaction(k) .

Sub-phase 2: In this phase we consider a Node pair j and i such that Node j has received a message sent from node i. The contents of the message play a big role in what follows in this phase. If the information in the message is new relative to node j, in that case this message is forwarded to all of j's fault free neighbors. If the information is the same, the message is discarded since the node has already received such a message before. If the information is old relative to the node j's information in that case, the message is loaded with the new content and sent back to only the sender of the message, that is node i. This is represented as follows:

- New information: If the content is relatively new, then the message data and local data changes are made. That is, $\forall$i , if msg.event[i] >$event_j[i]$, $event_j[i]$ is updated to msg.event[i] and if, $event_j[i]$ ¿msg.event[i] , msg.event[i] is updated to eventj[i].
- Old information: If it is old, then the message information is changed using the local information in the node and this message is sent out to node i. The update is made as $\forall$i , if, $event_j[i]$ ¿ msg.event[i], msg.event[i] = $event_j[i]$.
- Same information: The message is discarded if the content is the same as that of local node j's.

Before this sub-phase ends, node j compares checks its tester's event counter value. In case it is odd it would imply node j is a fault free orphan, in which case it initiates a step where it sends a testme request to one of its fault free neighbors at random.

3. **Sub-phase 3:** When a node k has propagated a message to a node j, which

doesn't reply with an acknowledgment this implies that node j has turned faulty before node k has noticed it. Since node k would have only sent a message to node j if it was fault free with respect to it implies the tester of node j has failed to record the change and propagate it, in other words the failure of node j's tester has occurred. Thus node k now updates its event counter of node j to indicate the failure and then disseminates the information in the network.

This is clearly shown by the figure below, where k forwards the message it has received from node x to node y and node j, node y replies with an acknowledgement but node j times out. Thus node k creates a message and sends it to its fault free neighbors x and y.

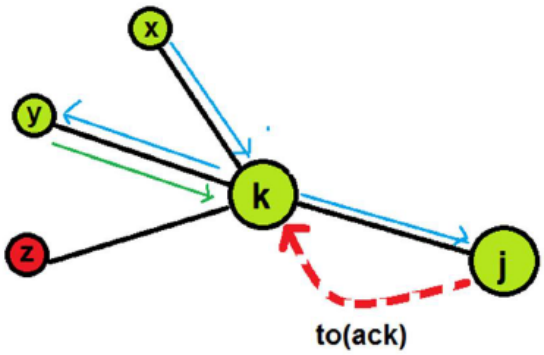


Figure 3.7: Sub-phase 3

Thus after the study of this phase we now know two definite ways to detect failures in a network which are through periodic testing in the testing phase and sub-phase 3.

Sub-phase 4: This sub-phase depicts the actions that take place when a node rejoins a network after failure. According to the theory established so far, a newly rejoined node only has an idea of its physical neighbors, in other words it keeps no information about the status (faulty/ fault free) of its neighbors. Thus when it rejoins it lacks a tester. It at random sends a request to one of its physical neighbors without knowing whether it is faulty or fault free. If it

meets a fault free node then it has a new tester else it tries till it finds a fault free node. This is depicted in the figure below, where node in pink is the newly joined node. After node k in the figure has accepted the request to become

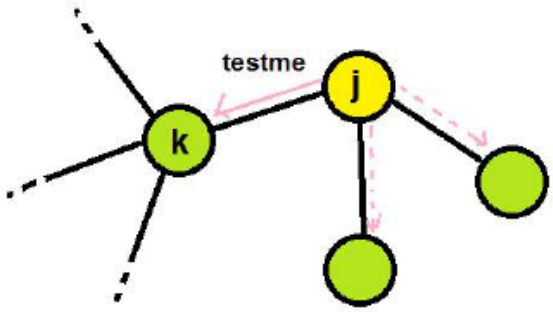


Figure 3.8: Sub-phase 4

j's tester node k will create a message containing j's arrival and circulate it in the rest of the network. Thus the node j will ultimately know the status of all nodes when this message reaches it.

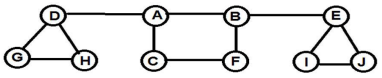
Chapter 4

Work Done

4.1 Non-partitionablity

Due to failure of a node if a network partitions in two or more connected component then that node is called an articulation point. While running the algorithm there can be more than two articulation points and hence it is important to point out those point which causes the partition in the network. Although the distributed algorithm continuously runs in different connected components of network topology but due to partition each connected component remains away from each other and wouldn't be able to communicate to each other unless the failure node is repaired. If at time t1 the network partitions into connected component due to failure of a node and remains disconnected till time t2 then in the time interval t2 - t1 the diagnosis of the algorithm would not be possible as the nodes in the network are unable to know the status of other nodes and hence don't return any solution. Therefore to point out those nodes which failure causes the partitioning of the network following algorithm is used. A network topology is represented by graph G = (N, L), where N is set of nodes n_1, n_2. . . n_n, and L is the set of links. If removal of node n in G results in a disconnected graph, then n is called an articulation point. If removal of node n causes the partition of network, there exist distinct link l_1 and l_2 such that

n is in every path from l_1 to l_2.



Node(i)	DFN(i)	L(i)	Articulation
A	1	1	Y
B	2	2	Y
C	7	1	N
D	8	8	Y
E	3	3	Y
F	6	1	N
G	9	8	N
H	10	8	N
I	4	3	N
J	5	3	N

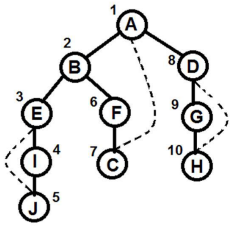


Figure 4.1: An example to find articulation points in a random topology of 10 nodes.

In first step, the network topology is traversed using depth first search. Each traversed node is given a depth first number respectively in the order they were traversed and then back edging was done from respective node to corresponding nodes which were connected in the network topology. Finally least value (where least value of a node is the least numbered dfn node that can be obtained by a back edge from the node to one of its ancestors or the least node that can be obtained by a back-edge from any one of its descendants) is compared to depth firs number. If depth first number for node i is less than the least vale of node i then the nodes causes partitionablity of network topology else not. When network partitions due to failure of one or more node(s), algorithm return the respective node(s) which causes the partition of the topology network.

4.2 Modified Algorithm

The following changes were made to the existing algorithm. Every node now will consist of an extra array of size N, where N is the number of nodes in the system. This array will consist of three values-0,1,2. Every message sent in the system will have an extra bit added to the end of it. The value of this bit will either be 0 or 1 depending on:

- 0, if it is currently not testing any other node.
- 1, If it is currently testing some other node.

Thus when the message arrives at a node, the receiver stores the message. It then compares the last bit of the message with the corresponding bit of that sender with the bit in the Test array. If there is a mismatch in that case the test array is overwritten with current value of bit. Thus after receiving the message the node knows which of its neighbors are testing some other node and which are not testing other nodes at that moment in time. Now when a fault free orphan is created i.e a node which does not have a tester at the moment, will check its test array first. It will check the test array to see if there exists any node which has a 0 entry in its corresponding test array location. If yes, then it checks to see how many such nodes are there which have 0 as entry, the first node which appears in linear search will then be sent a request for adding the fault free node as its tester. In case, there is no node which has a 1 as entry in that case, then a request is sent to any node at random. Thus the overall will result in the case fault-free requests made will reduce and thus the overall number of messages passed which will be a deciding metric in the performance of the algorithm.

Election Algorithm: The following priority based election algorithm is used for the case when a fault-free orphan tries to obtain a tester or a repaired node awakens. It has a 2 level priority-high and low. The high priority is given to that neighbor isn't currently testing any other node, as indicated by the presence of a 0 in the test array of that node. This is so, because the tester can immediately be assigned a tester as this neighbor is free to provide its testing services. The low priority is when all nodes in test array have 1s in which case a request is sent at random to the neighbors for tester. The pseudo-code for the same is provided below:

1. Check the test array to see if a there is a 0, if yes send that node a request.
2. Else send a request at random to any of the nodes.

3. Repeat above steps until the testing graph is formed.

Extra processing at each node: A fault free orphan is a node which currently doesn't have a tester. There are two ways to determine whether a node is a fault free orphan or not. In both cases we need to perform the extra processing as mentioned.

1. A node when receives a message from some other node checks to see if it's tester has failed. It does so by first obtaining which is it's tester from the Testnb array and then checking for an odd value at in the message that has arrived for that particular node. The presence of Odd value ensures, it's tester has failed and thus it has become a fault free orphan.

2. A node could become a fault free orphan when it's tester node has changed its state from testing it to some other node because it received a Testme message from some other fault free orphan.

In this case, the node which was testing it, will send a Carry Message to all of its neighbors including the node itself. The Carry message is designed to let the node know that the sender is testing someone else, this is ensured by the very fact that Carry messages to all neighbors except the neighbor which the tester will be testing. Once the node receives such a Carry Message, it goes to its Testnb array and changes the content to indicate that the tester is no longer testing it. Thus the node concludes that it has now become a fault free orphan- that is a node which doesn't have a tester at the moment.

Chapter 5

Simulation and Results

5.1 Simulation

The simulation was performed in Java environment.The following is the input to the system:

- Randomly chosen 50 nodes topology network.
- Events (faults/repair) occurring in the network after each testing round chosen randomly.

The following is output of the system: The output of the algorithm is shown after each testing round. Initially we have considered a fully fault free network. Thus event counter values of each of the nodes are all initially zero. After the random faults are introduced these event counter values will change.

1. Event counter values of each node.
2. The dissemination latency.
3. Number of messages passed in the testing round.
4. Testing graph for the next round.

The randomly chosen input to the network is shown as follows:

Failure	11
Failure	40
Failure	8
Failure	28
Repair	8
Repair	28
Failure	41
Failure	27
Failure	33
Failure	30
Repair	11
Failure	4
Repair	30
Failure	35
Failure	10

Figure 5.1: Events Occurred

5.2 Results

As we can see in the fig5.2, for the first fault, that is failure of node 11 the event counter value of node 11 is 1(odd) indicating that it is faulty. Since the remaining nodes are fault free, and hence their event counter is 0(even).

The dessimination latency that is the minimum time required for the information to reach the farthest node in the network from the initiator of the message is displayed in fig5.3. Also the number of messages passed in the network is shown.

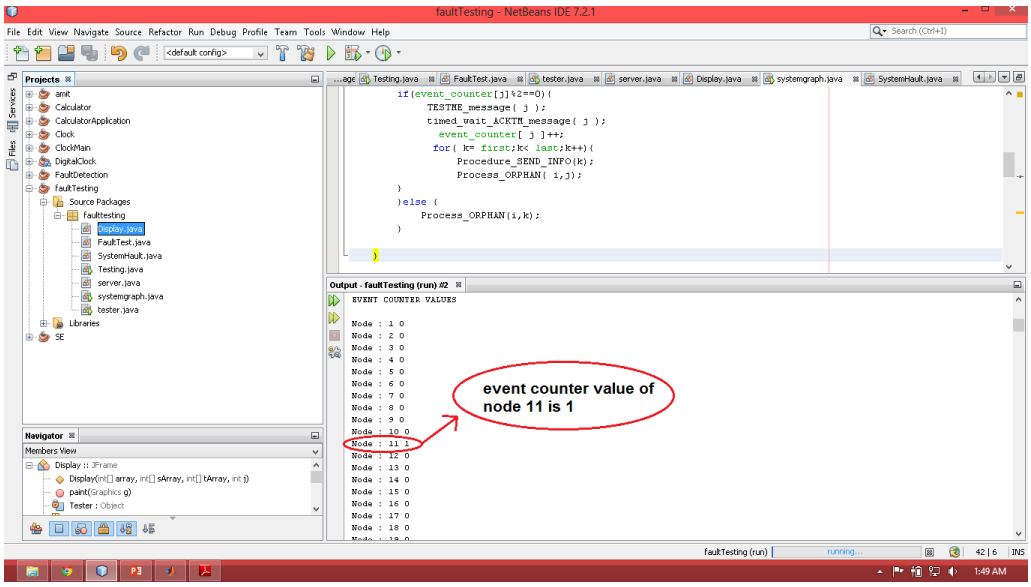


Figure 5.2: simulation run for the first event occurring in the network

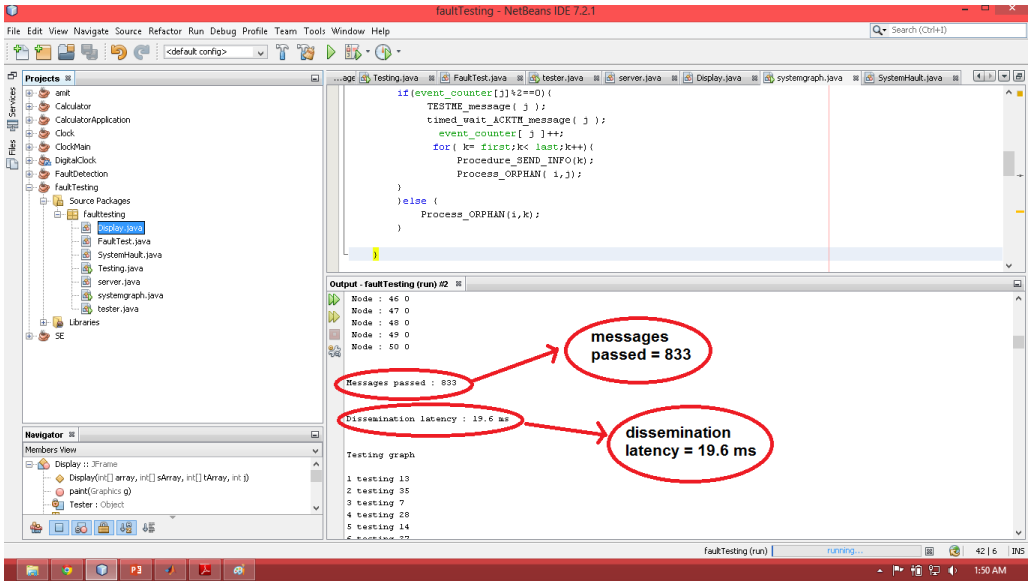


Figure 5.3: number of messages passed and the dissemination latency

A comparative plot between the modified algorithm and existing algorithm is shown in fig5.4. This plot shows that for all points expect 2, the number of messages passed in the modified algorithm is lower than that of the existing algorithm.

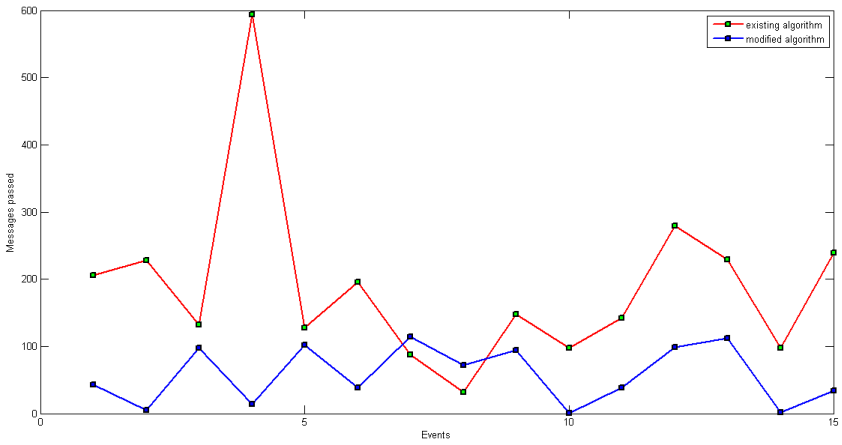


Figure 5.4: A plot of the number of messages passed vs the events occurred in the network.

A plot of the dissemination latency vs events (fig5.5) occuring in the network is shown.

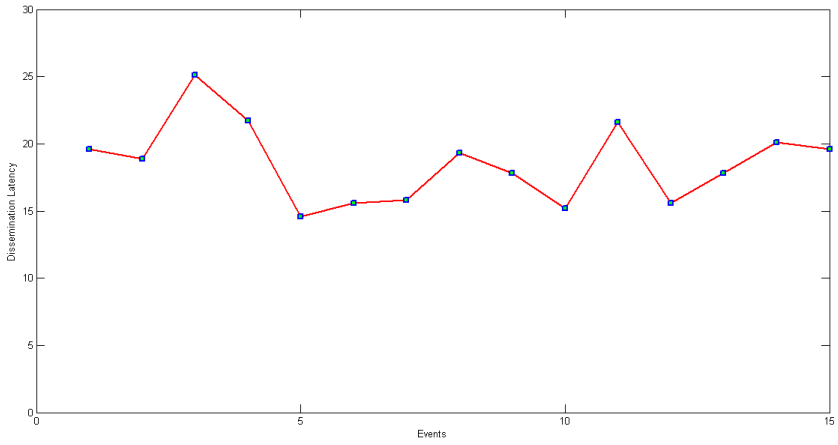


Figure 5.5: Plot of the dissemination latency vs events

Chapter 6

Conclusion

The system described in the above network can be effectively used to examine faults in general arbitrary networks. With the introduction of the modified algorithm, we see that the number of messages passed in the system will decrease in most cases. The overall advantages of the algorithm are that node failure can occur at any time in the network; this is important considering the random nature of faults that occur in practical cases. Secondly, the node that has just been repaired can directly enter the network without waiting for other nodes to detect it. Thirdly, nodes that contain faults do not have to be tested after a fixed period, thus help lowering the number of messages passed.

Chapter 7

References

1. M. Stahl, R. Buskens, and R. Bianchini, "Simulation of the Adapt On-Line Diagnosis Algorithm for General Topology Networks", Proc. IEEE 11th Symp. Reliable Distributed Systems, 1992.

2. M. Stahl, R. Buskens, and R. Bianchini, On-line diagnosis in general topology networks, in Proc. Workshop Fault-Tolerant Parallel and Distributed Systems, 1992.

3. Elias Procopio Duarte Jr. andrea weber, "Simulation of a Distributed Connectivity Algorithm for General Topology Networks", 2002.

4. Luiz Carlos Pessoa Albini, Elias Procpio Duarte Jr. Roverli Pereira Ziwich, "A Generalized Model for Distributed Comparison-Based System-Level Diagnosis", Journel of the brazilian computer society, 2011.

5. L. P. Saikia, K. Hemachandran, "Simulation of System Level Diagnosis in Distributed Arbitrary Network", Journal of Theoretical and Applied Information Technology, 2007.

6. Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery" ACM Transactions on Computer Systems, pp.398-461, 2002.

7. Dorothy E. Denning, "An intrusion-detection model", IEEE Transactions on Software Engineering, pp.222-232, 1987.

8. Leslie Lamport, "The part-time parliament", ACM Transactions on Computer Systems, pp.133-169, 1998.

9. Andreas Haeberlen, Petr Kuznetsov, "The Fault Detection Problem", 2008.

10. Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel, "PeerReview: Practical Accountability for Distributed Systems", In Proceedings of the 21st ACM Symposium on Operating Systems Principles pp.175-188, 2007.

11. M. Panda, P. M. Khilar, "Distributed Soft Fault Detection Algorithm in Wireless Sensor Networks using Statistical Test", 2nd IEEE International Conference on Parallel, Distributed and Grid Computing, 2012.

12. A. Mahapatra, P. M. Khilar, "Fault Diagnosis in Wireless Sensor Network: A servey, IEEE Communications and Tutorials, Issue 99, pp.1-27, April 2013.

13. M. N. Sahoo, P. M. Khilar, "Survivialability of IEEE 802.11 Wireless Lan Against Access Point Failure,International Journals of Applications in Engineering,Technology and Sciences, vol 1, No 2, pp.424-428, 2009.